Networking from an OS Perspective

CS6410 Lecture 12

Robert Escriva

Cornell CS 6410, October 7, 2010

Van Jacobson

- Known for his work on the TCP/IP stack (this paper in-particular).
- `traceroute`, `pathchar`, `tcpdump`.
- Currently at PARC. [*]

[*] `http://www.parc.com/about/people/88/van-jacobson.html`

Two papers in one?

This paper has a strange feel as if the figures (and their respective captions) are disconnected from the main body, which cites other papers to support the algorithms. (Also, half the paper is appendices)

Is this a good or bad thing?

Two Key Algorithms

slow-start This ensures that any individual node will not saturate the network with a flood of packets on a new connection.

congestion avoidance This takes over when a connection is established to try continuously squeezing more out of the network.

Self-Clocking Connections

TODO: Insert figure 1.
Packet spacing slows to match the smallest pipe on the path.

The slow start algorithm

Straight from the paper:

- Add a *congestion window*, `cwnd`, to the per-connection state.
- When starting or restarting after a loss, set `cwnd` to one packet.
- On each ack for new data, increase `cwnd` by one packet.
- When sending, send the minimum of the receiver's advertised window and `cwnd`. [**?**]

Getting to Equillibrium

- Let R be the round-trip-time (RTT)
- Let W be the window size (measured in packets)
- $Rlog_2W$ time to open the the window.

Behavior Without Slow-Start

TODO: Insert Figure 3

Behavior With Slow-Start

TODO: Insert Figure 4

Exponential Backoff

Exponential backoff is the only scheme that really works.

Retransmit Timers (According to RFC793)

- $R = \alpha R + (1 - \alpha)M$.
- Retransmit timeout interval: $rto = \beta R$.
- Suggested $\beta = 2$.
  - Adapts to loads of at most 30%.
  - High loads cause retransmission of delayed packets.

RFC793 Retransmit Timers

TODO: Insert Figure 5

Improving rtt mean and variation

$$Err = M - A$$

$$A \leftarrow A + gErr$$

$$D \leftarrow D + g(|Err| - D)$$

- A is average of RTT.
- D is mean deviation of RTT.
- g is gain.
- E_r is random error.
- E_e is estimation error.

Code

$$2^n A \leftarrow 2^n A + Err$$

$$g = \frac{1}{8}$$

```
M -= (SA >> 3);
SA += M;
if (M < 0)
    M = -M;
M -= (SD >> 3);
SD += M;
rto = ((SA >> 2) + SD) >> 1;
```

Mean+Variance Retransmit Timers

TODO: Insert Figure 6

Congestion Avoidance Algorithm

- Congestion $\Rightarrow$ packet loss (typically).
- Packet lost $\Rightarrow$ timeout (typically).
- Implies that timeout can be used to measure congestion.

Adapting to congestion

$W_i = dW_{i-1} \qquad (d < 1)$

- Exponential decrease for persisted congestion.

$W_i = W_{i-1} + u \qquad (u \ll W_{max})$

- Additive increase for absence of congestion.

The slow start algorithm

Straight from the paper:

- On any timeout, set `cwnd` to half the current window size (multiplicative decrease).
- On each ACK for new data, increase `cwnd` by $1/\texttt{cwnd}$ (additive increase).
- When sending, send the minimum of the receiver's advertised window and `cwnd`. [**?**]

Experiments

TODO: Insert Figure 7

No Congestion Avoidance

TODO: Insert Figure 8

Congestion Avoidance

TODO: Insert Figure 9

Overall Improvement

TODO: Insert Figure 10/11

Proposed Future Work

- Look to the gateways to control congestion too.
- Look at second-order feedback for the increments used.

Takeaway points

- Simple observations can make behavior more predictable.
  - Reasoning about packet loss and using it as a way to measure congestion.
- Feedback-based algorithms are generally good.

Authors

- Stefan Savage - Professor at UCSD.
- Neal Cardwell - Seems to have dropped from radar (or he's been a graduate student researcher for 10+ years).
- David Wetherall - Associate Professor at Washington.
- Tom Anderson - Robert E. Dinning Professor at Washington.

The Big Picture

TCP’s congestion control assumes everybody “does the right thing.”

ACK Division

TODO: Figure 1.

DupACK Spoofing

TODO: Figure 2

Optimistic ACKing

TODO: Figure 3

Optimistic ACKing

This attack doesn’t preserve reliability, and is the most clever of the three.

TCP Daytona

- Built on the TCP implementation of Linux 2.2.10.
- Intended to abuse other systems using the three attacks described.

The best line in the paper

Needless to say, our implementation is intentionally not “stable”, and would likely lead to congestion collapse if it were widely deployed.

Applicability

TODO: Table 1
Many vendors are vulnerable. A bug within NT saves it from DupACK Spoofing (are there exploitable side-effects of this bug?).

Solutions

- ACK Division
  - Have the congestion-control mechanism operate solely on segment/byte granularity.
- DupACK Spoofing
  - Add a nonce to the sent segments and ACKs.
- Optimistic ACKing.
  - Sum the nonce numbers. Send sum on in-sequence data; send the nonce itself for out-of-sequence data.
  - Randomly change segment boundaries.

Cumulative Nonce

TODO: Figure 7.

Key Points

- Cooperative protocols do not always stand up when clients misbehave.
- When everyone else is playing nice, it's easy to take advantage of the situation.
- Reinforces the point of the previous paper (more work should be done at the gateways).